8. Heng

Program Name: Heng.java Input File: heng.dat

Heng just learned about CRC (cyclic redundancy check) in networking class and has looked up what Wikipedia says about it, which is:

> **"A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents."**

Essentially, when a message is sent, two other things are sent: a key, and a checksum. The receiver performs an algorithm on the message using the key and determines if the result of that algorithm matches the checksum that is sent.

Heng decides to find a simple way of doing this, just for practice, and discovers this simple algorithm. First he starts with a binary string represented by the hex value 098, which represents some data, and decides on a key value of 5. The equivalent binary strings for these two hex values are:

000010011000 and 0101

The process he discovers uses long division modulo 2 in binary, but instead of subtracting it uses the bitwise XOR process to reduce the message. When XOR is applied in a bitwise manner, if the two bits being considered are the same, the resulting bit is zero, otherwise the result is 1. The process for the data example given would look like this:

000010010000 divided by **0101**

```
000010011000
    101
    00111
      101
      0100
       101
       00100
         101
          01
```

He carefully notes that only the remainder after the XOR process is important, and that the final remainder should be expressed using the number of significant digits of the key, minus 1.
In the example to the right, since the key, 0101, has three significant digits (the leading zero does not count), the final remainder must be expressed in two SD, or 01.

Input - Several pairs of hex values, each pair on one line, separated by a single space.

Output - For each pair of input values, calculate and output the final checksum based on the algorithm described and demonstrated above.

Sample data:

```
098 5
F 3
AC 5
71C A
8E6B 6
```

Sample Output:

```
01
0
11
000
01
```